13 FIRST and FOLLOW Sets

Wednesday, October 23, 2024 12:07 PM

Motivation: Guide the algorithm:

$A \to B \mid C \mid d$

parse_A()

use the token to decide → parse_B() / parse_C()

- if we know what terminals a B can begin with and what terminals a C can begin with.

then

compare the token with these sets of terminals to make a decision.

- FIRST Sets:

A set of termals a derivation from a non-terminal can begin with

$$FIRST(\alpha) = \{a \mid \alpha \overset{*}{\Rightarrow} a\beta\}$$

Example:

$S \to AB \mid BA$

$A \to aB \mid cS$

$B \to bc$

$FIRST(a) = \{a\}$

$FIRST(b) = \{b\}$

$FIRST(c) = \{c\}$

$FIRST(B) = \{b\}$

$FIRST(A) = \{a, c\}$

$FIRST(S) = \{a, b, c\}$

$A \to aB$

$A \to cS$

$S \to AB \to aBB$

$S \to BA \to bcA$

$S \to AB \to cSB$

Algorithm:

1) if X is a terminal FIRST(X) = {X}

- Repeat:

2) if $X \to \lambda$ then add λ to FIRST(X)

3) if $X \to Y_1 Y_2 Y_3 \ldots Y_k$ then

a) if for some i, $\lambda \in FIRST(Y_j)$ for $j < i$ and $a \in FIRST(Y_i)$ then add a to FIRST(X)

b) if for all i, $1 \le i \le k$, $\lambda \in FIRST(Y_i)$ then add λ to FIRST(X)

E.G.

$S \to ABCD$
$D \to AB$
$A \to a \mid \lambda$
$B \to b \mid \lambda$
$C \to c$

$FIRST(A) = \{a\ \lambda\}$
$FIRST(B) = \{b\ \lambda\}$
$FIRST(C) = \{c\}$
$FIRST(D) = \{a, b\ \lambda\}$

$FIRST(a) = \{a\}$
$FIRST(b) = \{b\}$
$FIRST(c) = \{c\}$

$FIRST(S) = \{a\ b\ c\}$

D	D	D
AB	AB	AB
aB	B	B
	b	λ

$S \to ABCD$
$X \to Y_1 Y_2 Y_3 Y_4$

E.G.

$E \to TE'$
$E' \to +TE' \mid \lambda$
$T \to FT'$
$T' \to *FT' \mid \lambda$
$F \to int \mid (E)$

	FIRST	
E	int	(
E'	+	λ
T	int	(
T'	*	λ
F	int	(

- Using FIRST Sets:

To guide our parser

$A \to B \mid C \mid d$

```
FUNCTION parse_A()
  IF token in FIRST(B) THEN
    parse_B()
  ELSIF token in FIRST(C) THEN
```

of course: we are assuming

$FIRST(B) \cap FIRST(c) = \emptyset$
$FIRST(B) \cap FIRST(d) = \emptyset$

```
    IF token in FIRST(B) THEN
      parse_B()
    ELSIF token in FIRST(C) THEN
      parse_C()
    ELSIF token = 'd' THEN
      get_token()
    END
  END.
```

$FIRST(B) \cap FIRST(c) = \emptyset$
$FIRST(B) \cap FIRST(d) = \emptyset$
$FIRST(c) \cap FIRST(d) = \emptyset$

So:
1. Compute first set
2. check for "pairwise disjointness"

fix by either read more tokens, or change the grammar.

E.G.

$A \to dB \mid dC \mid d$
$B \to ax$
$C \to ag$

⇓ left factoring

$A \to dA'$
$A' \to B \mid C \mid \epsilon$
$B \to ax$
$C \to ag$

⇓ factor common elements

$A \to dA'$
$A' \to aA'' \mid \epsilon$
$A'' \to B' \mid C'$
$B' \to x$
$C' \to y$

- FOLLOW Sets:

Intuition: for a non-terminal, the terminals that can appear immediately to its right in a derivation

eg.

eg.

Suppose: $S \xrightarrow{*} ABC \xrightarrow{*} AbcC$ so $b \in FOLLOW(A)$

Definition:

$FOLLOW(A) = \{a \mid S \xrightarrow{*} \delta A \beta \xrightarrow{*} \delta A a \Gamma\}$

Note: $\$ \in FOLLOW(S)$ where S is the start symbol

Example:

$S \to AB \mid BA$
$A \to aB \mid cS$
$B \to bc$

$FOLLOW(S) = \{\$, b\}$
$FOLLOW(A) = \{b, \$\}$
$FOLLOW(B) = \{a, c, b, \$\}$

$S \to AB \to Abc$
$S \to BA \to BaB$
$S \to BA \to BcS$

$S \to AB \to cSB \to cSbc$
$S \to AB \to aBB \to aBbc$

$S\$ \to AB\$$ $S\$ \to BA\$$

Algorithm:

// to compute follow sets

1) add $ to FOLLOW(S) where S is the start symbol

REPEAT:

2) IF $A \to \alpha B \beta$ THEN
add everything in FIRST(β) to FOLLOW(B)

3) IF $A \to \alpha B$ (or $A \to \alpha B \Gamma$ and $\lambda \in FIRST(\Gamma)$) THEN
add everything in FOLLOW(A) to FOLLOW(B)

Note: λ is never in a FOLLOW set.

explain Rule 3)

$S \to Ax \to \alpha Bx$

$S \to Ax \to \alpha B\Gamma x \to \alpha Bx$

E.G.

$S \to ABCD$
$D \to AB$
$A \to a \mid \lambda$
$B \to b \mid \lambda$
$C \to c$

	FIRST	FOLLOW
S	a b c	$
D	a b λ	$
A	a λ	b c $
B	b λ	c $
C	c	a b $

E.G.

- $E \to TE'$
  $E' \to +TE' \mid \lambda$
  $T \to FT'$
  $T' \to *FT' \mid \lambda$
  $F \to int \mid (E)$

$(E) \to (TE')$

	FIRST	FOLLOW
E	int (	$)
E'	+ λ	$)
T	int (	+) $
T'	* λ	+) $
F	int (	* +) $

USES:

- Error recovery on a recursive descent parser.

Intuition.

$A \to \alpha$	FIRST(A)
parse_A()	FOLLOW(A).

Pseudocode:

if we want to call parse_A()
- Read tokens until token is in FIRST(A)
- parse_A()

- Read tokens until token is in FIRST(A)
- parse_A()
- Read tokens until token is in FOLLOW(A)

E.G.

```
FUNCTION old_parse_A()
...
...
```

```
FUNCTION new_parse_A()
  WHILE token not in FIRST(A) DO
    error(token)
    getToken()

  old_parse_A()

  WHILE token not in FOLLOW(A) DO
    error(token)
    getToken()
```

```
// version 2: handle a missing A
FUNCTION new_parse_A()
  WHILE token not in (FIRST(A) union FOLLOW(A)) DO
     error(token)
     getToken()

  IF token in FIRST(A)
     old_parse_A()

  WHILE token not in FOLLOW(A) DO
     error(token)
     getToken()
```